

User manual

D-11805001-ADeS-UM-05a

ADeS
v0.3.0

September 2007

HISTORY

Date	Vers.	§	Description	Author
2007-02-02	01	all	New user manual for ADeS v0.2.2	Axlog Ingénierie
2007-03-23	02	§3, §4.1, §4.8	Updates for ADeS v0.2.3	Axlog Ingénierie
2007-03-29	03	§2	Required JVM version	Axlog Ingénierie
2007-06-25	04	§2 §2, §3 §3.5, §3.6, §3.7 §4.3 §4.8	Versions of required software components Updates of illustrations references Updates for ADeS v0.2.4 Details about <i>zero snapshot</i> file Simplification of the trace file description	Axlog Ingénierie
2007-07-14	04c	§2 §2.1	EMF available on Callisto Update Site ADeS dependencies are not latest versions Screenshots update	Axlog Ingénierie
2007-09-27	05a	§B.1 §2 §3.4, §4.3	Corrections on the example Versions of required software components Simulation editor use and description	Axlog Ingénierie

TABLE OF CONTENTS

1. PREFACE	9
1.1. Prerequisites	9
1.2. In this manual	9
1.3. Naming conventions.....	10
1.4. For more information	10
1.4.1. Technical support.....	10
1.4.2. Web site	10
2. INSTALLATION	11
2.1. Installation through the update site	11
2.2. Installation of the archive plug-in	15
2.3. Installation of the standalone archive.....	15
3. BASIC TUTORIAL	17
3.1. Step 1: Launch of Eclipse.....	17
3.2. Step 2: Modeling an AADL architecture	19
3.3. Step 3: Simulation project creation	21
3.4. Step 4: Simulation creation	25
3.5. Step 5: Instantiation of the simulation.....	27
3.6. Step 6: Loading of the simulation	29
3.7. Step 7: Execution of the simulation	31
3.7.1. AADL instances view	32
3.7.2. AADL instance attributes view.....	33
3.7.3. ADeS console view.....	33

3.7.4. Jimex event manager view	34
4. CONCEPTS	35
4.1. Simulation project	35
4.2. Simulation builder	35
4.3. Simulation editor	35
4.3.1. Overview page	35
4.3.2. Information page	37
4.3.3. Source page	37
4.4. Simulation instantiation	37
4.4.1. Zero snapshot file generation	37
4.5. Simulation loading	37
4.6. Simulation execution	37
4.7. Graphical representation	38
4.8. Trace of the results	38
4.9. Time management	38
5. TASKS	39
5.1. Adding a new scheduling policy	39
5.2. Adding a new graphical view	39
6. REFERENCE	41
6.1. Processor behavior	41
6.2. Thread behavior	41
6.3. ADeS behavior annex	41
6.3.1. Content of the annex	41
6.3.2. Example	42
ANNEXE A. GLOSSARY	45

A.1. Generic terms	45
A.2. Terms specific to the project	45
ANNEXE B. AADL EXAMPLES	47
B.1. AADL description of the demo system.....	47

1. PREFACE

AADL (Architecture analysis & design language) provides a means to describe real time embedded systems, with both their software aspects and their execution platform aspects, and the relations between the components constituting it. Specifying a system in AADL instead of in any natural language makes possible many advanced techniques to help in the development life cycle: iterative modeling, analysis, code generation, reusability, etc.

It is often interesting to estimate what will be the behavior of a system under development without waiting for its definitive form. For this purpose, simulation approaches are very useful, and take naturally place in the development life cycle. Depending on the evolution of the system under development, a simulation may be used on its specification, refined during the design phase, or compared with the result of the development at the end of the project.

ADeS aims at simulating the behavior of such system described with AADL. It is developed as an Eclipse plug-in, in relation with Osate, the AADL modeling plug-in developed by SEI, and Topcased, which provides a graphical modeling tool.

Attention! The version of ADeS which is presented here (version 0.2.4) is still under development. Several useful features are not yet integrated. This manual also contains sections which will be completed in the future.

1.1. Prerequisites

You are supposed to know AADL before reading this manual. For more information about this language, please visit its official web site: <http://www.aadl.info>, or contact us for support, training or expertise.

1.2. In this manual

This manual is organized as follow.

Chapter 2 explains the installation of ADeS.

Chapter 3 proposes a short tutorial to use the simulator.

Chapter 4 presents the main simulation concepts used by ADeS.

Chapter 5 explains how to perform specific tasks to go further with this simulator.

Chapter 6 gives some general references.

1.3. Naming conventions

This manual uses the following conventions:

- Command lines appear in `typewriter` fontface.

1.4. For more information

1.4.1. Technical support

For technical support or to ask for new features, please contact Axlog.

Axlog Ingénierie
19-21 rue du 8 mai 1945
94110 Arcueil, FRANCE

Tel: +33 1 41 24 31 00

Fax: +33 1 41 24 07 36

E-mail: aadl@axlog.fr

1.4.2. Web site

A web page dedicated to ADeS is available on the Axlog web site. Its URL is subject to modifications in the future. In such a case, please visit the Axlog web site.

http://www.axlog.fr/aadl/ades_en.html

2. INSTALLATION

ADeS may be provided under several forms: source code, binary plug-in for Eclipse, standalone application encompassing all the required dependencies...

This version of ADeS requires the following software components:

- JRE 1.6 (<http://www.java.com/en/download>)
- Eclipse 3.3 Europa (<http://www.eclipse.org>)
- EMF 2.3 (Europa Discovery Site)
- GEF 3.3 (Europa Discovery Site)
- Osate 1.5 (<http://la.sei.cmu.edu/aadlinfosite/OSATEUpdateSite/>)

The following software components are recommended:

- Topcased 0.11.0 (<http://www.topcased.org>).

2.1. Installation through the update site

ADeS may be installed through an Eclipse update site. Its URL is:

<http://www.axlog.fr/ades/update-site>.

In order to install ADeS and the pre-required software tools specified above (ADeS dependencies), select “*Help | Software Updates | Find and Install...*”. The “*Install / Update*” dialog opens. Select “*Search for new features to install*” and click on the “*Next*” button.

Create a new remote site for ADeS by clicking on “*New Remote Site...*”. And fill fields as illustrated by Illustration 1.

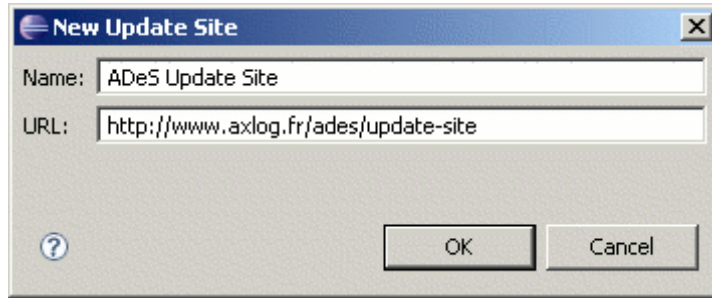


Illustration 1: ADeS update-site creation

Then create one remote site for OSATE. Its update site URLs is specified above between brackets.

Select the three remote sites (ADeS, Europa, and OSATE) as shown in the Illustration 2 and click on the “*Finish*” button.

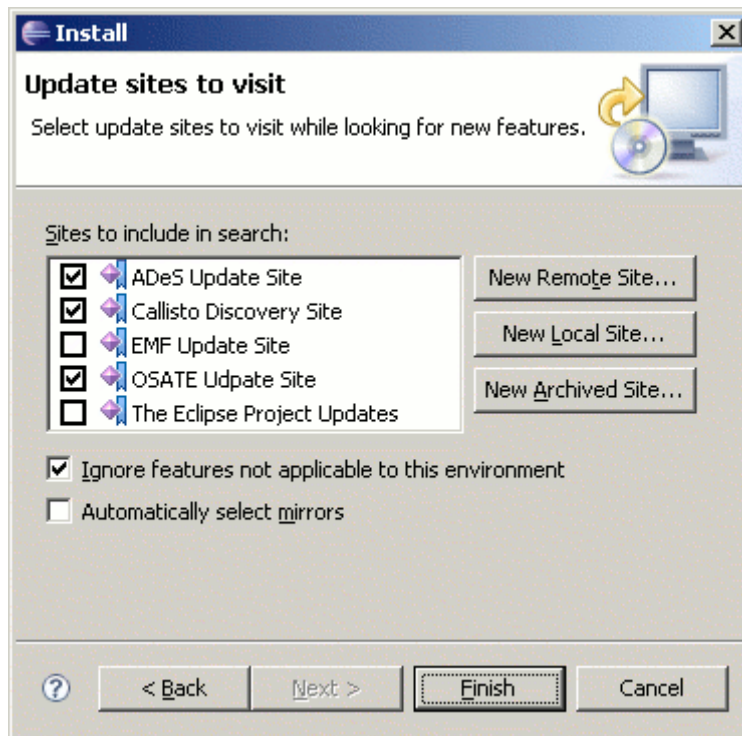


Illustration 2 Install dialog

Nota : Eclipse may ask you to choose the update site mirrors: choose the default ones works fine.

In the “*Updates*” dialog (see Illustration 3), check boxes so that:

- “*Show the latest version of a feature only*” is not selected;
- “*Filter features included in other features on the list*” is selected.

Develop each update site in order to see the content of each update site (see Illustration 3). Develop “ADeS” and select “ADeS 0.3.0”: an error message should appear and traduce a missing dependency. In order to correct that, click on the “Select Required” button.

Nota : Click on “Select Required” without having developed each update site should give nothing.

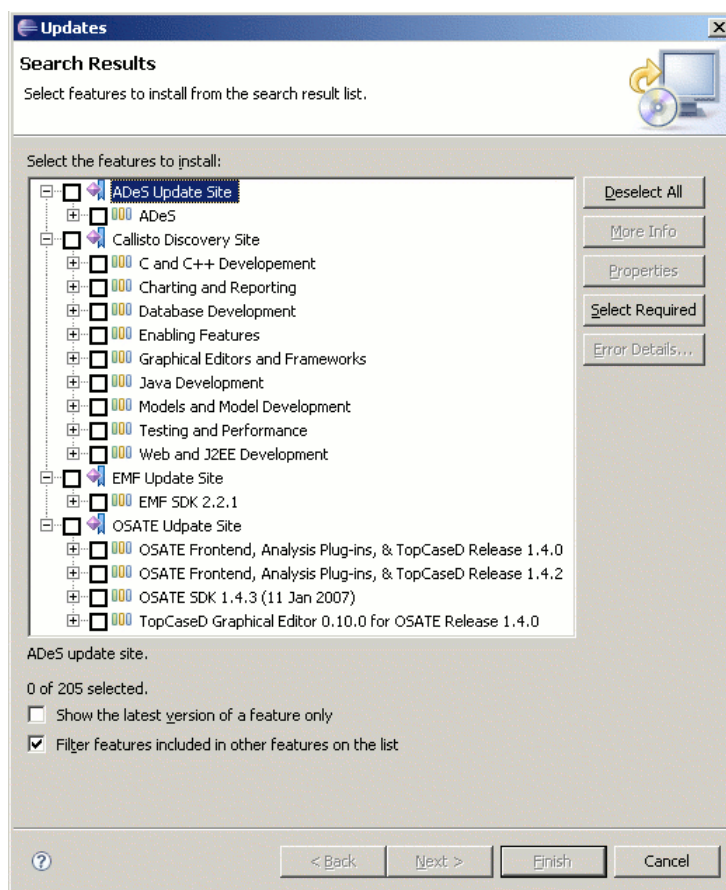


Illustration 3: Update site installation (1/2)

Nota : The version of OSATE that is selected by default is the latest one. This latest version of OSATE is compatible with ADeS v0.2.4 because it requires a version of EMF which is not compatible with ADeS to. User should manually select “Open Source AADL Tool Environment 1.4.9”.

If the computation of the dependencies has worked fine, the features selection should be similar to the one contained in Illustration 4 – versions excluded.

If this is the case, click on the “Next” button. Perform the end of the installation wizard and launch download and installation processes of the selected features.

Once installation is over, a restart of the Eclipse SDK is needed. After restarting, the Eclipse environment and ADeS are ready to be used.

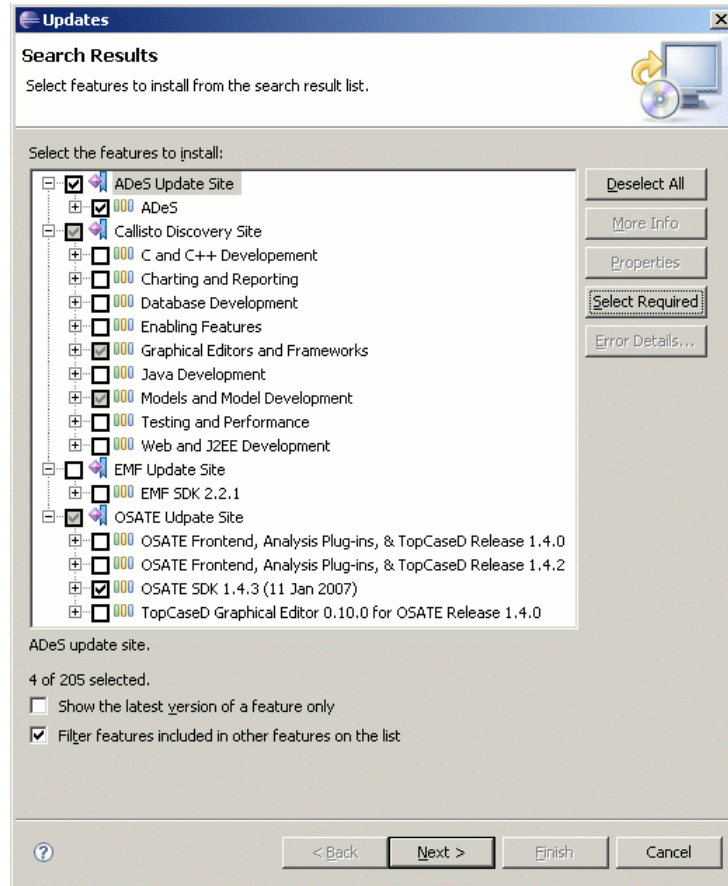


Illustration 4: Update site installation (2/2)

In order to be sure that ADeS is correctly installed, select “*Help | About Eclipse SDK*”. The ADeS icon should appear in the *About Eclipse SDK* dialog (see Illustration 5).

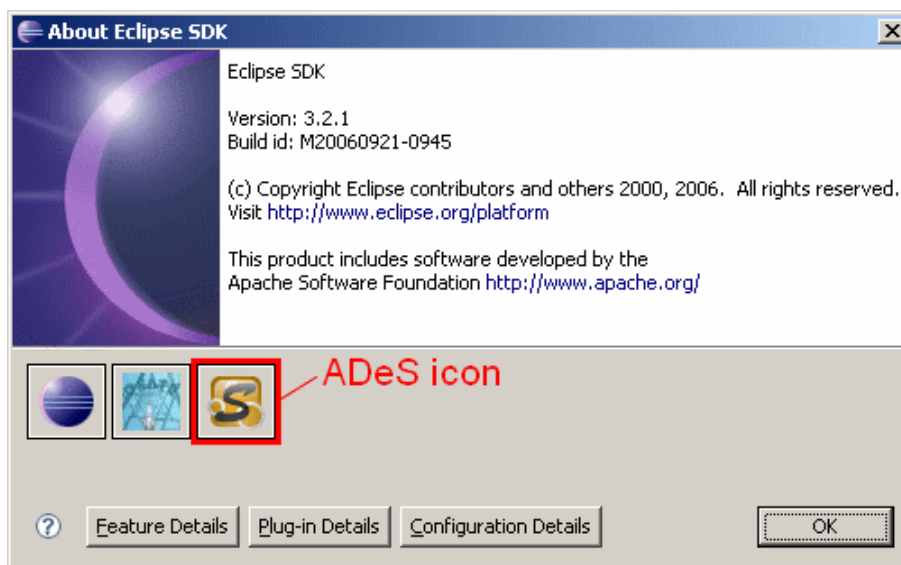


Illustration 5: About Eclipse SDK dialog

2.2. Installation of the archive plug-in

ADeS may be provided as an archive file (.zip) containing the binary files of the plug-in.

To install it, make sure to have the pre-required software tools specified above, and just unzip the ADeS installation file into the Eclipse installation directory.

2.3. Installation of the standalone archive

ADeS may also be provided as an archive file containing all what is needed to run the program: Eclipse, the required plug-ins, and the ADeS plug-in. To install it, just unzip the installation file where you want and run `eclipse`.

3. BASIC TUTORIAL

This section presents a short tutorial to teach the use of ADeS. This tutorial is based on the modeling and simulation of a simple one-processor system with two threads which periodically communicate through ports.

3.1. Step 1: Launch of Eclipse

Start Eclipse.

If Eclipse starts for the first time, or if the option is not disabled, a dialog appears to choose the workspace (Illustration 6).

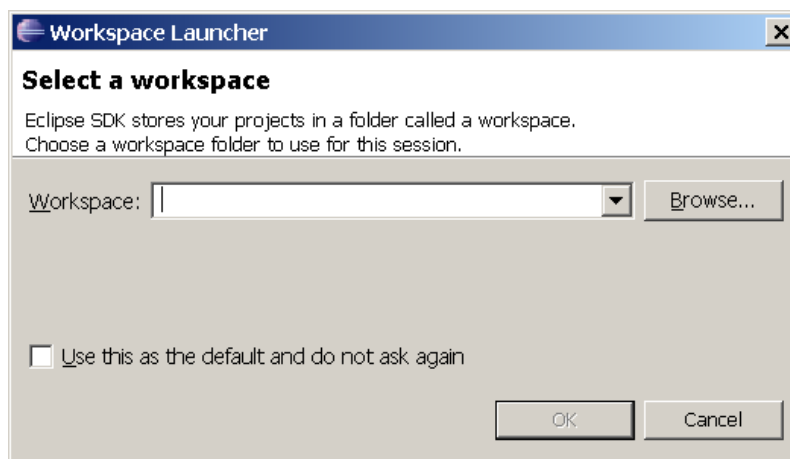


Illustration 6: Dialog for selection of the workspace

If Eclipse starts for the first time, it opens a welcome screen (Illustration 7). Close this screen by clicking on the cross of the Welcome tab.

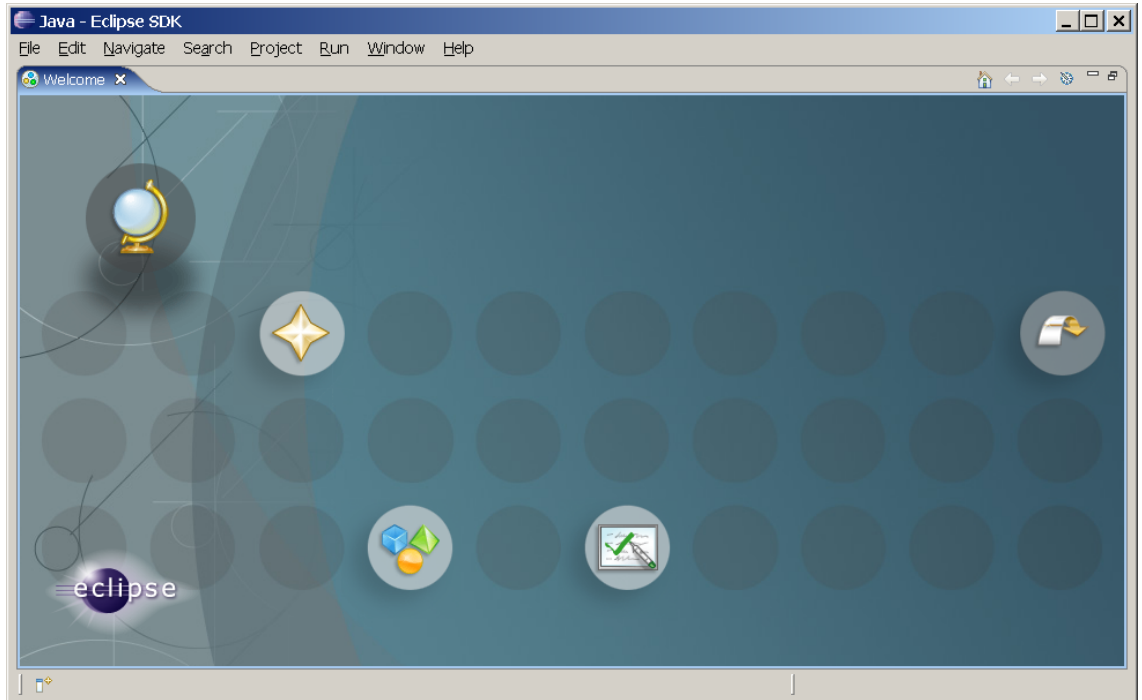


Illustration 7: Welcome screen of Eclipse

Now we have to open the AADL perspective to edit the description of the system. To open this perspective choose the menu “Window/Open Perspective/Other...”. A dialog is open to select the desired perspective (Illustration 8). Choose “AADL”.

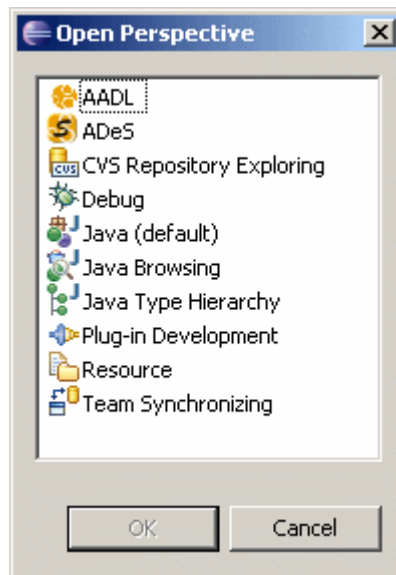


Illustration 8: Open perspective dialog box

We get an empty Eclipse workspace (Illustration 9).

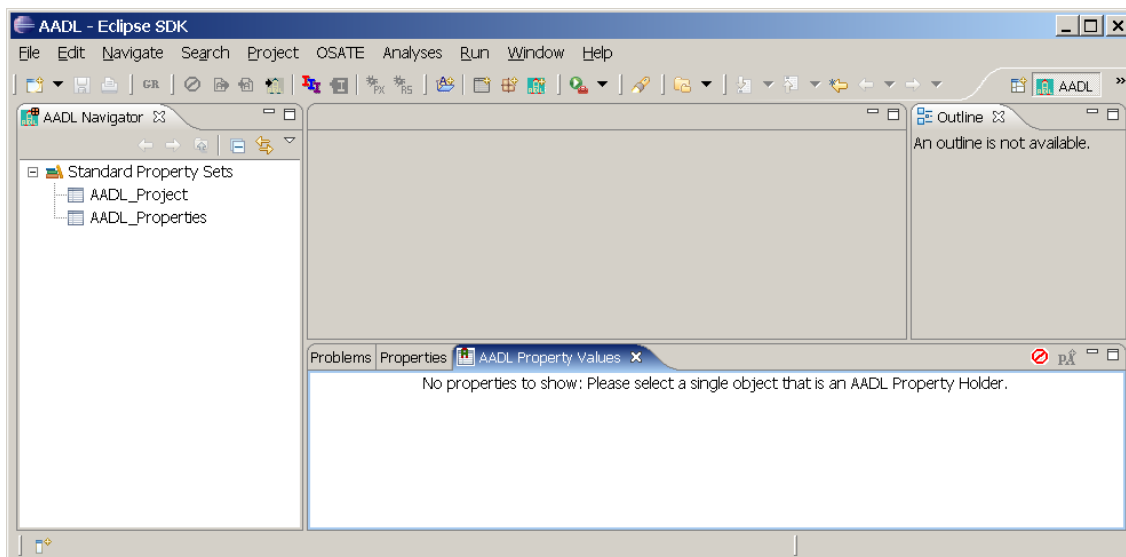


Illustration 9: Eclipse framework with an empty AADL workspace

3.2. Step 2: Modeling an AADL architecture

This step consists in producing the AADL description of a simple system with one processor executing two threads. The first one is periodic and triggers the second one through an event port.

Create a new AADL project: Select menu “File/New/Project...”. A dialog box appears (Illustration 10). Select “Aadl Project”, click on the “Next >” button and give a project name (e.g., “demo.aadl.project”).

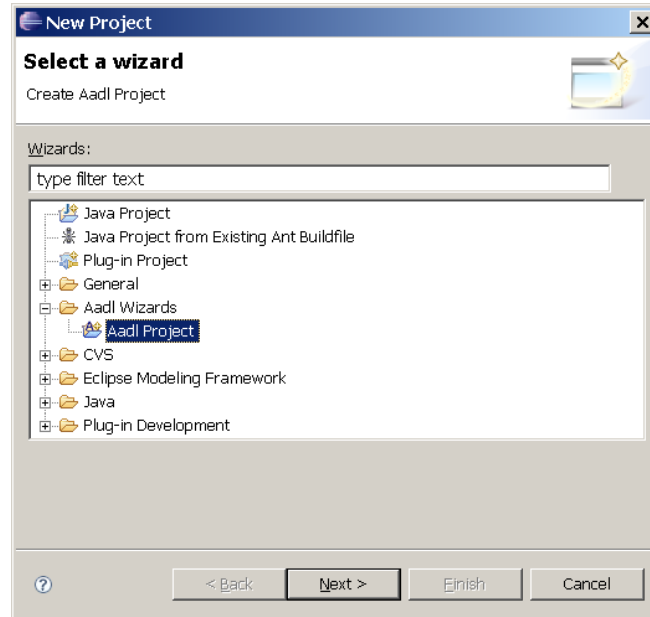


Illustration 10: Dialog box for the AADL project creation

Once the project is created, it appears in the “AADL Navigator” view. It contains two directories: “*aadl*” for the textual AADL description files, and “*aaxl*” for the XML AADL description files. For more information, please read the Osate user manuals.

In order to create a new AADL model, select the “*aadl*” directory encapsulated in the “*demo.aadl.project*” project, right-click on it and select “New/Other...” in the contextual menu. A dialog box appears (Illustration 11). Select “*Aadl Model*”, click on the “Next >” button, select “*Aadl Spec*”, “*Text File*”, and enter a name for the model (e.g., “*simpleSystem*”). A new empty file called “*simpleSystem.aadl*” is created. It should be also opened in the editor. If not, just double-click on it in the “AADL Navigator” view.

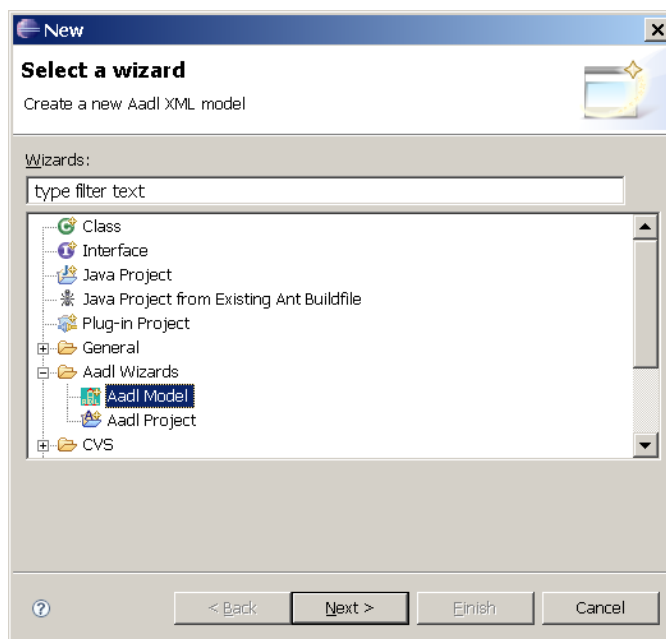


Illustration 11: Dialog box for the AADL model creation

In the editor you can edit your AADL model. The aim of this tutorial is not to learn AADL. A complete AADL model is proposed in annex of this document. Just copy and paste it in the AADL textual editor.

If the Topcased plug-in is installed, it is also possible to graphically edit the AADL model.

3.3. Step 3: Simulation project creation

Once an AADL project is created, a simulation project can be created. In order to do this, select menu “*File / New / Project...*”.

A dialog box appears (Illustration 12). Select “*ADeS Project*” in the “*ADeS*” wizards category, and click on the “*Next >*” button. It is strictly equivalent to use the contextual menu of “*AADL navigator*” (Illustration 13).

Then the ADeS project creation wizard starts (Illustration 14). On the first page on this wizard, fill the name of ADeS project to create (e.g. “*demo.ades.simulation*”) and click on the “*Next >*” button. The second page lists AADL projects which are opened in the current workspace and that can be referenced by the ADeS project being created. Select the desired AADL project (“*demo.aadl.project*”) and click on the “*Finish*” button.

At the end of the ADeS project creation, the ADeS perspective is automatically opened if “*Yes*” is chosen in dialog box asking for this opening (Illustration 15).

Nota : Another way for opening the ADeS perspective is to use the menu: “Window | Open Perspective | Other...”; it opens a dialog in which ADeS perspective appears under the “ADeS” label (Illustration 8).

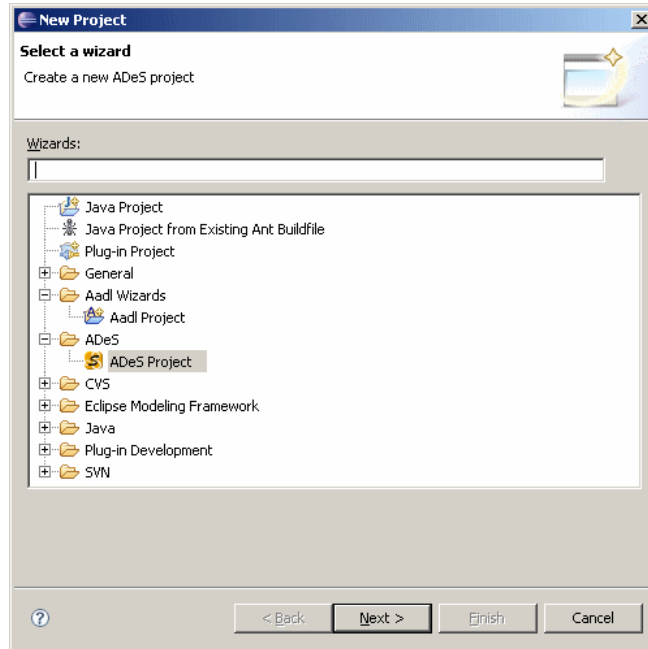


Illustration 12: ADeS simulation project creation (dialog box)

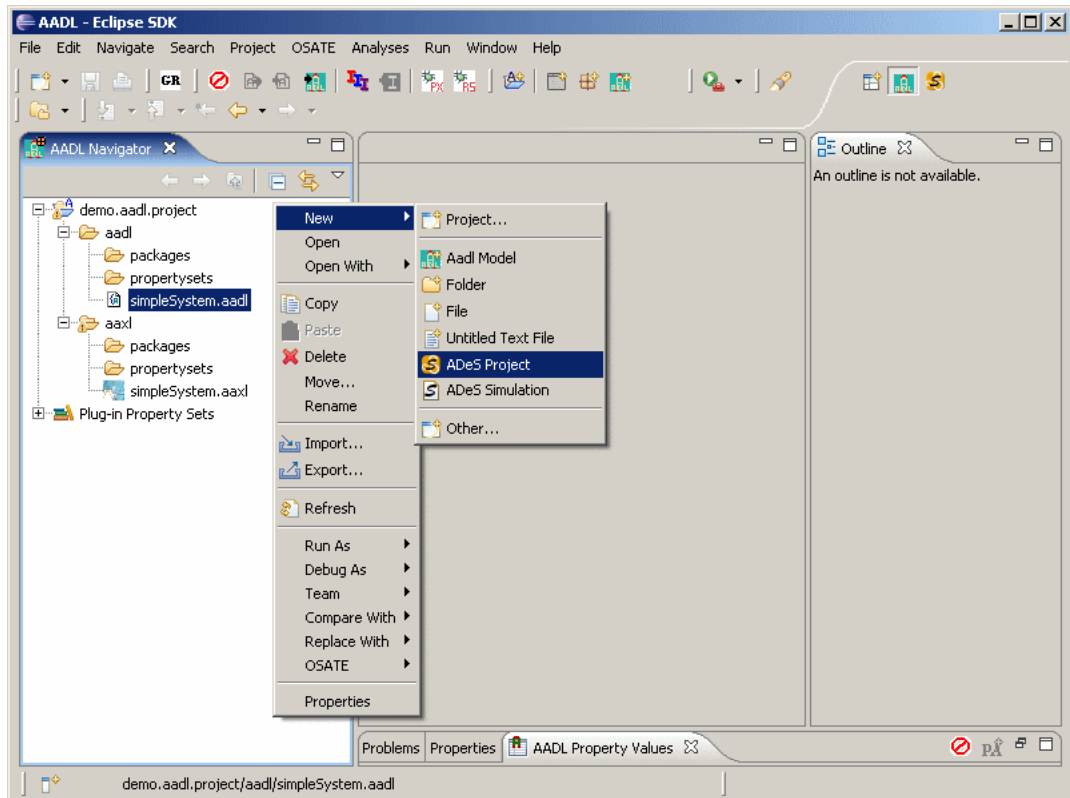


Illustration 13: ADeS simulation project creation (contextual menu)

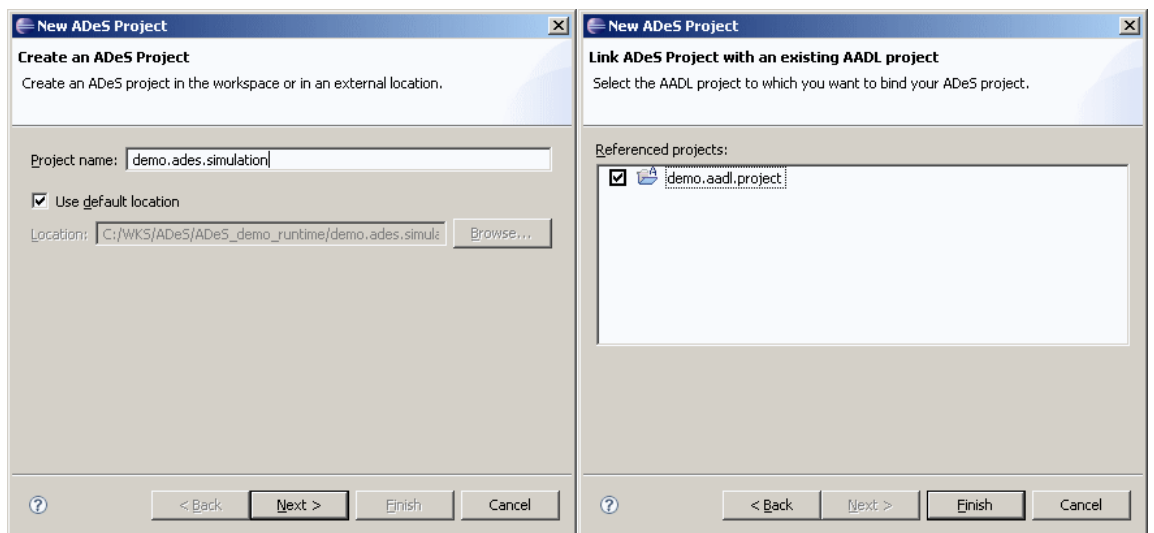


Illustration 14: ADeS simulation project creation wizard

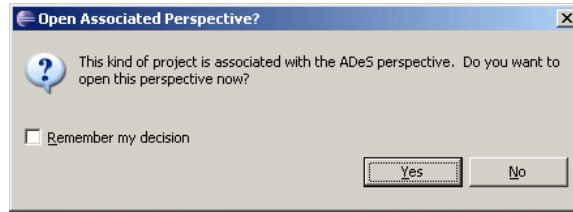


Illustration 15: ADeS perspective opening

Illustration 16 shows the opened ADeS perspective; both AADL and ADeS projects appear in the “Navigator” view.

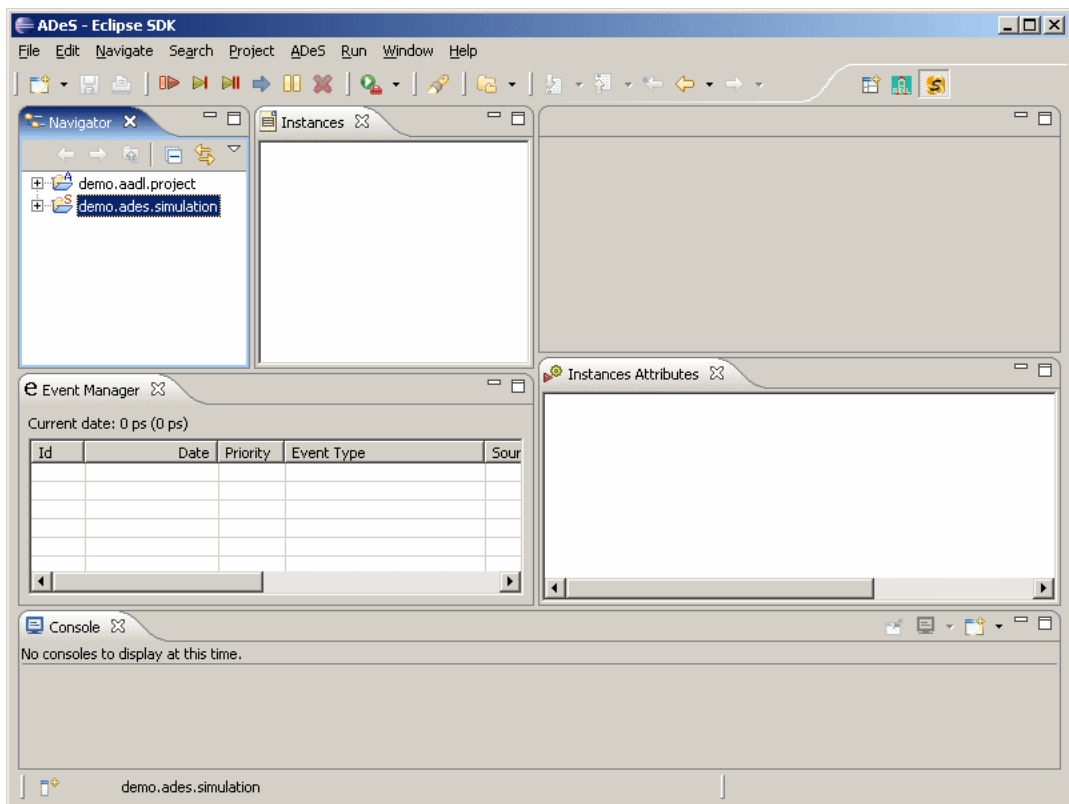


Illustration 16: ADeS perspective

The ADeS perspective proposes a set of views showing the system under simulation, its properties, and results of the simulation. It also proposes a tool bar to drive the simulation (Illustration 17).



Illustration 17: ADeS toolbar

Each button has a role in the control of the simulation. From left to right:

- start of the simulation;

- execution of one step of the simulation;
- execution of the simulation up to the end of the current time step;
- execution of the simulation up to a given date;
- stop of the simulation;
- reset of the simulation.

3.4. Step 4: Simulation creation

A dedicated wizard exists in order to create a new simulation. Select the ADeS project (“demo.ades.simulation”), launch the “ADeS Simulation Wizard” which is accessible from menu “File / New / Other...” in “ADeS” category (Illustration 18).

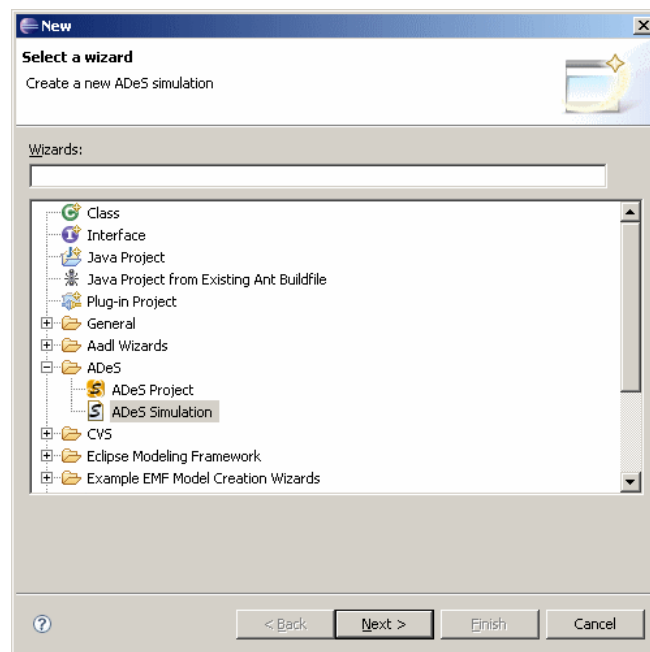


Illustration 18: ADeS simulation creation (dialog box)

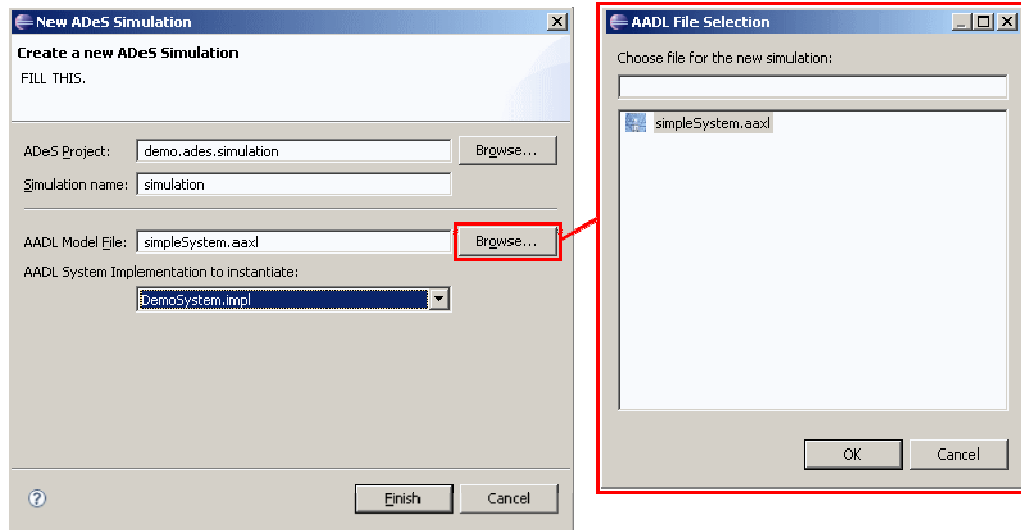


Illustration 19: ADeS simulation creation wizard

Fill text fields of the ADeS simulation creation wizard as shows Illustration 19 and click on “Finish”. A folder “simulation” containing a file “.ades” is created.

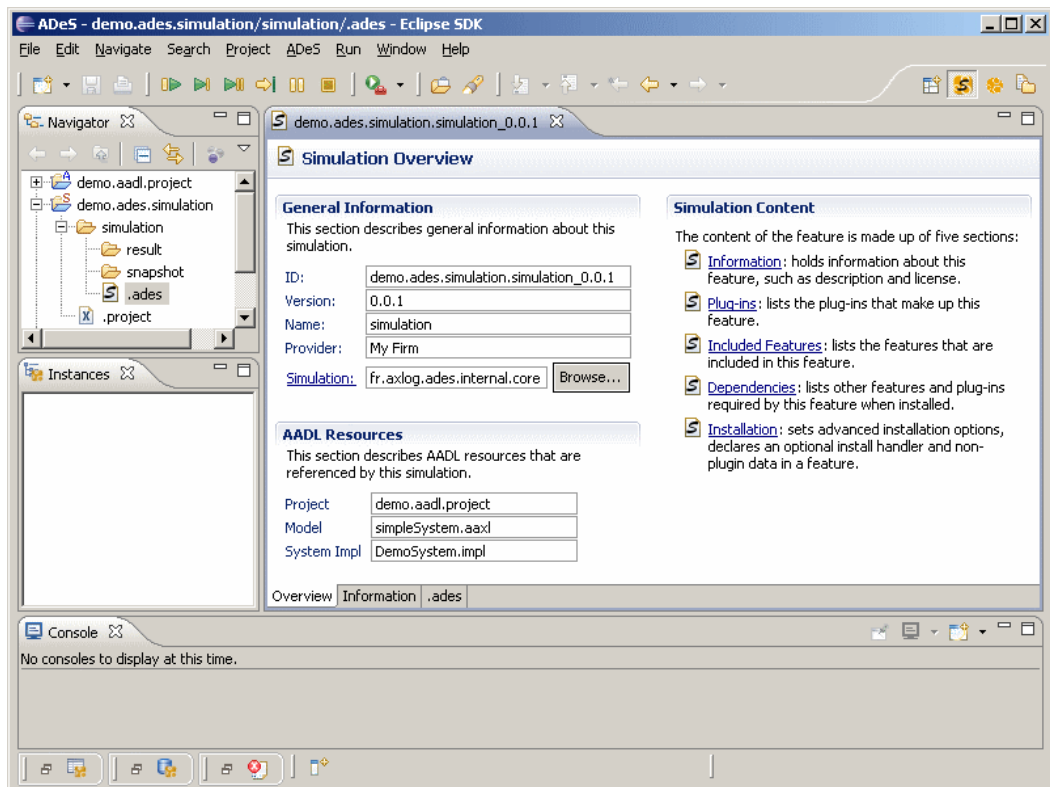


Illustration 20: ADeS simulation editor

In the same time, the simulation editor is opened (see Illustration 20), allowing the edition of the newly created “.ades” file. Fields filled in the simulation wizards appears inside this editor. As a consequence, it is possible to change them.

Nota : For the moment, the two references on the AADL model architecture or on the AADL system implementation can not be modified inside the simulation editor.

The AADL project field is not editable since this reference is associated to the parent simulation project.

3.5. Step 5: Instantiation of the simulation

It is now possible to instantiate the system implementation named `DemoSystem.impl` which is declared in the “*simpleSystem.aadl*” file.

The instantiation of the simulation is the phase during which simulation objects are created. In order to do this, ADeS gets from OSATE an instance of the system implementation chosen by the user and analyses it. The concerned system implementation is referenced by the new created simulation. Once simulation objects are created, a snapshot of the simulator is saved in a binary file “.snp”. This file is called the *zero snapshot* because it represents the state of the simulator at the date zero.

In order to instantiate the system implementation referenced by the simulation file (“.ades”), launch the instantiation action from the contextual menu “*ADeS | AADL instantiation*” of the navigator view (Illustration 21).

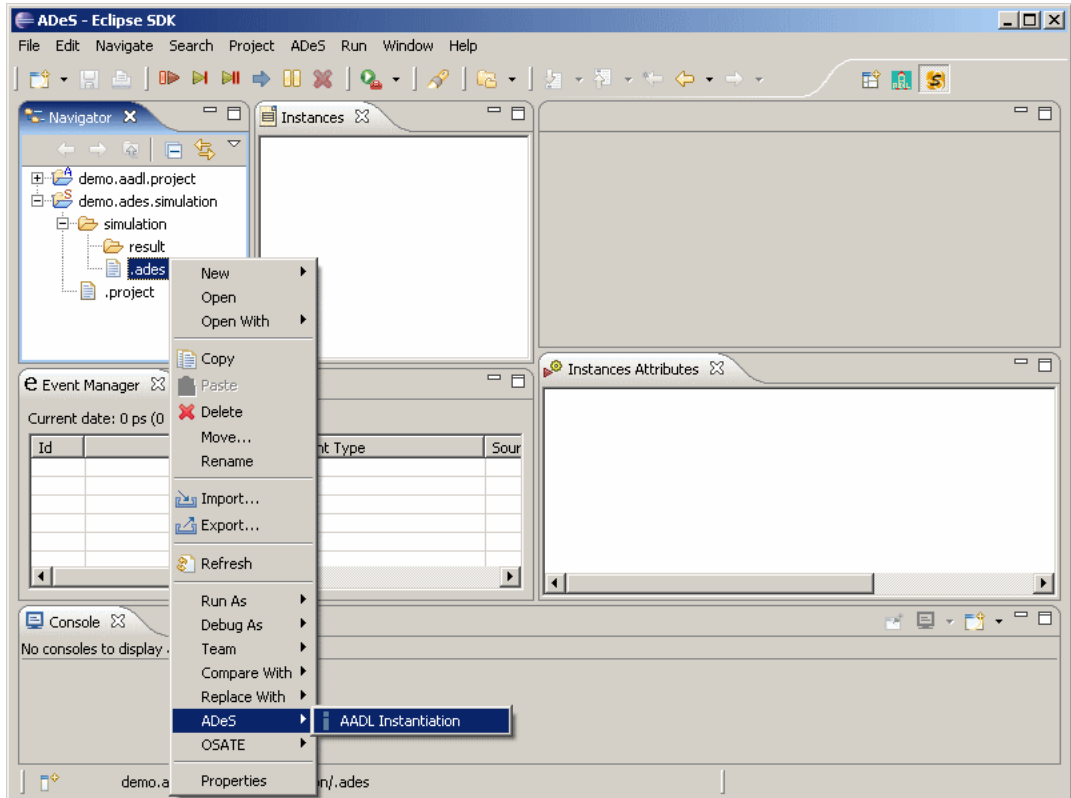


Illustration 21: AADL instantiation action

At the end of the instantiation action, the *zero snapshot* file is created under the “*snapshot*” folder of the simulation (Illustration 22).

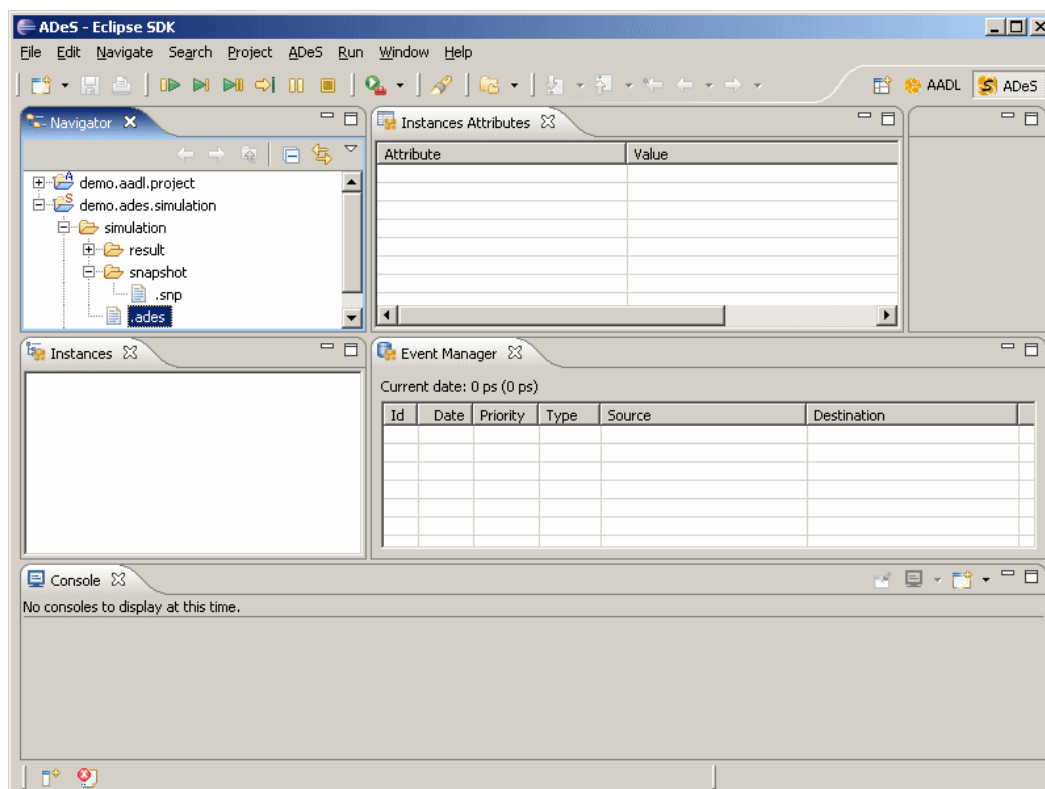


Illustration 22: ADeS perspective at the end of the instantiation

Nota : At this moment, the instance tree viewer contains no AADL instance.
From the simulation point of view, no instance exists yet.

3.6. Step 6: Loading of the simulation

A simulation can be executed if and only if the *zero snapshot* of that simulation has been loaded before.

In order to load a simulation represented by a *zero snapshot* file (“*.snp*”), launch the load snapshot action from the contextual menu “*ADeS | Load snapshot*” of the navigator view (Illustration 23).

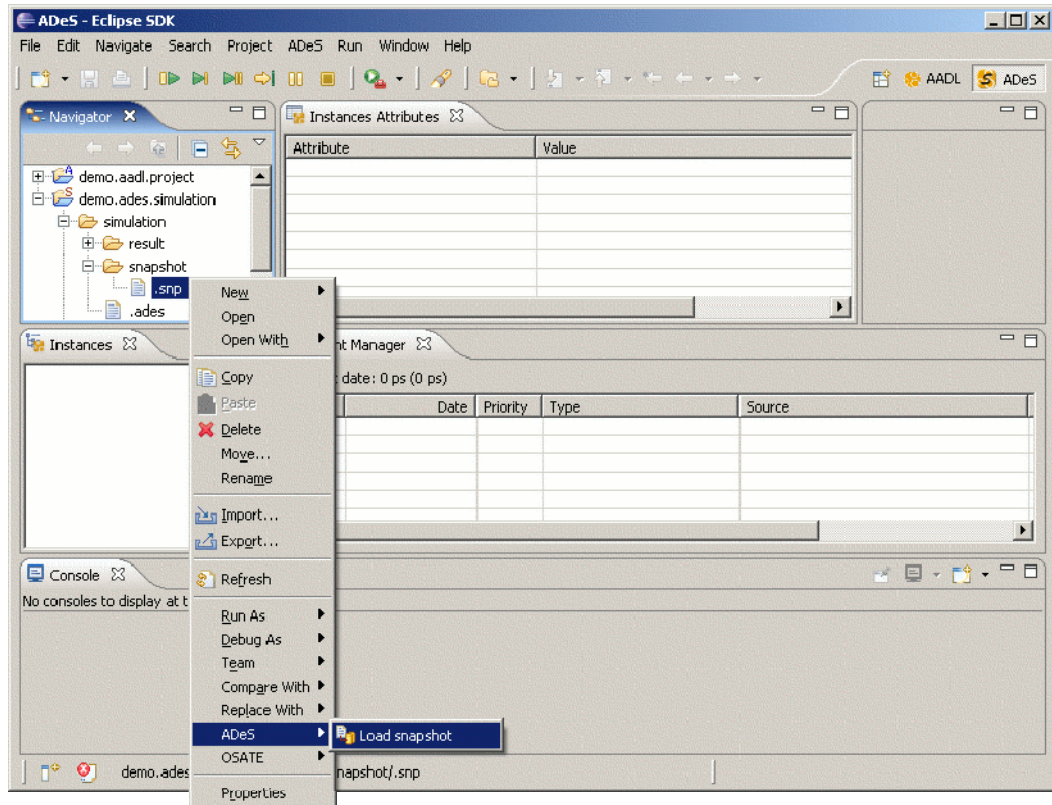


Illustration 23: Simulation snapshot loading

Once the snapshot of the selected simulation is loaded, the ADeS perspective is updated (Illustration 24): the instances components are displayed in the instances tree viewer (even if they are still inactive); the event manager view is refreshed.

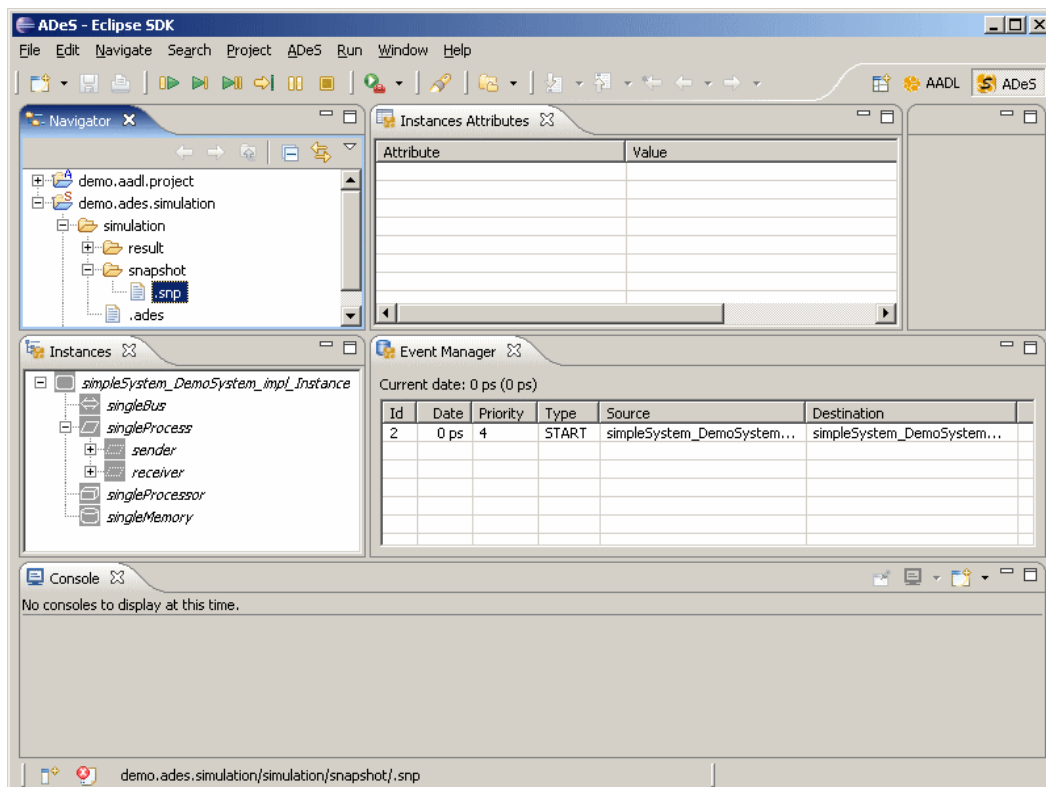


Illustration 24 ADeS perspective at the end of the snapshot loading

3.7. Step 7: Execution of the simulation

The simulation can now be launched (and controlled) thanks to the buttons of the ADeS toolbar (these actions are also available in the ADeS menu). The following table (see Table 3.1) summarizes the actions that can be used to control the simulation execution.

Icon	Label
	Run
	Next event step
	Next time step
	Goto
	Stop
	Reset

Table 3.1: Actions to use for the simulation execution control

Once simulation is launched, ADeS views are updated: display of the simulation results (Illustration 25).

There is still a lot of work for these graphical aspects. The current version does not contain every expected graphical components: statistic graphs, chronograms, etc.

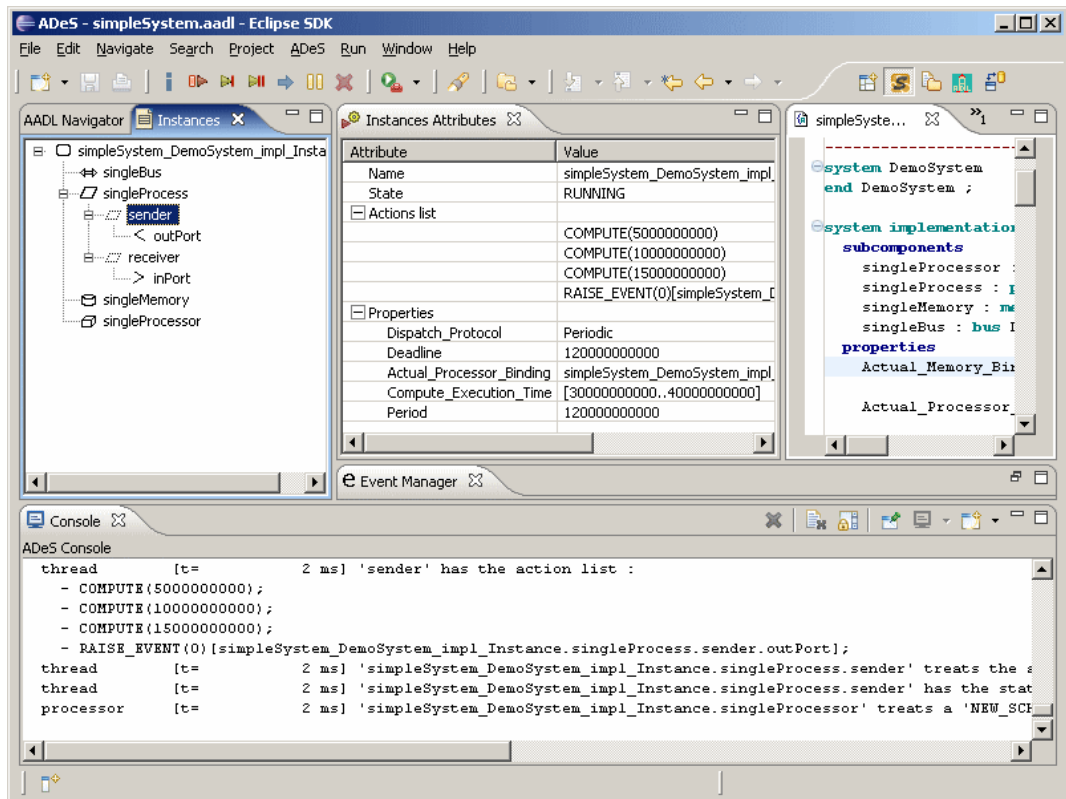


Illustration 25: ADeS perspective during the simulation

3.7.1. AADL instances view

This view contains a tree viewer into which simulated objects (AADL instances) are displayed. For the moment, simulated objects that can be displayed in that view are AADL component instances, AADL port instances and AADL access instances (bus access for example).

Inside the tree viewer, labels of instance components depend on their respective state:

- a white background with a normal font for active components;
- a grey background with an italic font for inactive components.

This is the main view of the ADeS perspective: the content of the other views belonging to the ADeS perspective depends on the selection in that instances tree viewer.

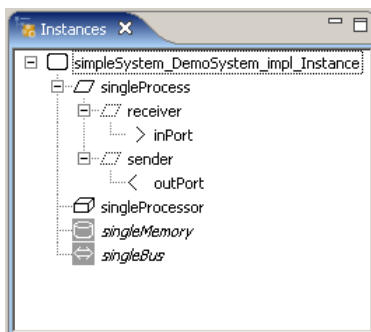


Illustration 26: AADL instances tree viewer

3.7.2. AADL instance attributes view

This view contains a table tree viewer into which attributes of the instance selected in the AADL instances tree viewer are displayed. According to type of that selected instance, this view is able to display: the name of the instance, its state, its actions list, its properties, etc.

This view listens the instance tree viewer selection, but it also listens the selected instance changes. For example in the case of a thread, if its internal state changes, the view will be updated with the new internal state of that thread.

Attribute	Value
Name	simpleSystem_DemoSystem_impl_Instance.singleProcess.sender
State	RUNNING
Actions list	compute(10 ms) compute(15 ms) raise(EVENT(0)[simpleSystem_DemoSystem_impl_Instance.singleProcess.sender.outPort])
Properties	Period: 120 ms Compute_Execution_Time: [30 ms..40 ms] Actual_Processor_Binding: simpleSystem_DemoSystem_impl_Instance.singleProcessor [STOPPED][default] Dispatch_Protocol: Periodic Deadline: 120 ms Synchronized_Component: true

Illustration 27: AADL instance attributes view

3.7.3. ADeS console view

The ADeS console view is based on Eclipse console view mechanism. It prints every simulation activity or instance change: internal state, subcomponents list, actions list, kernel events, problems, etc.

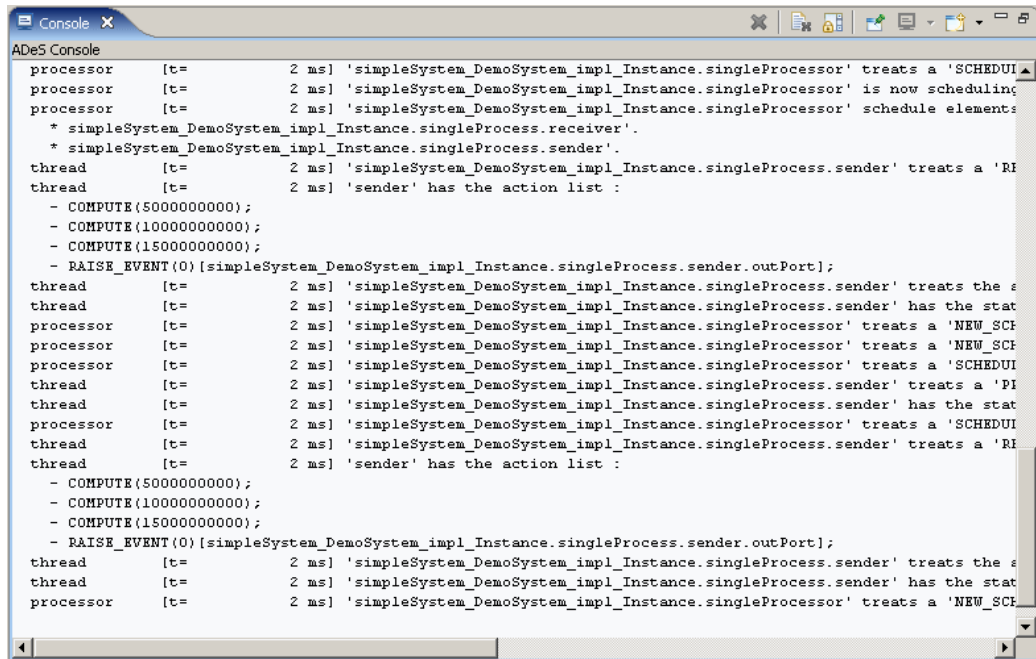


Illustration 28: ADeS console view

For the moment this console view is the best way for giving the simulation result. In the future, that console should probably be reserved to experts who would want to have a very precise return of what happened during the simulation.

3.7.4. Jimex event manager view

This view represents the events stack of the simulation kernel (called *Jimex*). As the graphical user interface of ADeS is still under development, such a view is very useful for seeing what exactly happens during simulation.

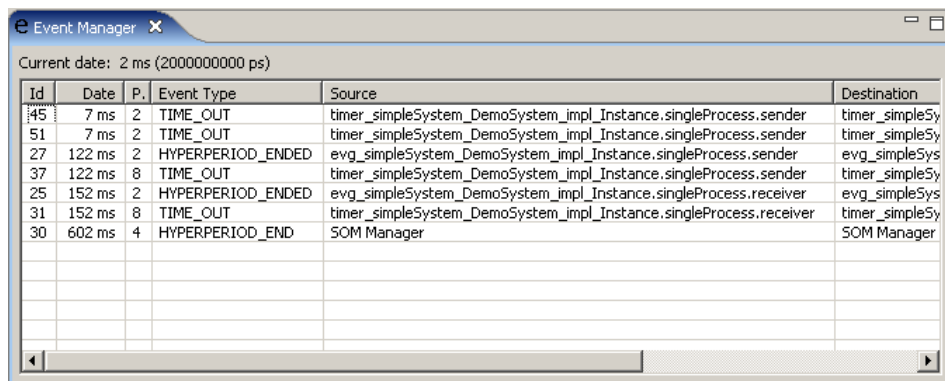


Illustration 29: Jimex event manager view

4. CONCEPTS

4.1. Simulation project

A simulation project is an Eclipse project which has the ADeS nature – to which the ADeS builder is associated.

A simulation project refers one and only one AADL project containing the AADL model encapsulating the system implementation to instantiate and to simulate.

A simulation project can contain several simulations. The simulated systems may be the same.

To be completed.

4.2. Simulation builder

To be completed.

4.3. Simulation editor

The ADeS simulation editor is an Eclipse-like multi-page editor which allows editing attributes and parameters of a simulation. The following paragraphs explain in detail the content of each page constituting this multi-page editor.

4.3.1. Overview page

The overview page of the ADeS editor represents an overview the simulation being edited. It contains general information such as the name, the version or the provider of the simulation. It also lists references on AADL resources: the AADL project that contains the AADL model that contains the AADL system that must be simulated.

4.3.1.a. General Information section

This section encapsulates fields providing the edition of simulation:

- name;
- version;
- ID;
- provider name;
- zero snapshot.

The simulation ID is built with respect to the following format:

`<simu-project-name>.<simu-name>_<simu-version>`

Nota : For the moment, a simulation name modification would have to impact on the workspace structure; whereas it should. Indeed, renaming a simulation should imply renaming the simulation container folder.

Nota : The simulation zero snapshot field is not fully supported yet. Next version of ADeS should bring Improvement on this.

To be completed.

4.3.1.b. AADL Resources section

This section encapsulates fields providing the edition of references on AADL:

- project;
- model;
- system implementation.

The *AADL project* whose name is displayed is the AADL project to which refers the parent simulation project.

The *AADL model* is the file that contains the description of the architecture that must be simulated.

The *AADL system implementation* is the component that must be instantiated in order to perform the simulation of the described architecture.

Nota : The AADL project field should not be editable since once a simulation is created inside the simulation project, it is supposed to simulate any architecture described in the AADL project referenced by its parent simulation project.

Nota : The support of the AADL model field is not complete: a Browse button should be added in order to choose the AADL model file in the list of available model files.

Nota : The support of the AADL system implementation is not complete either: this field should be a combo list.

4.3.1.c. Simulation Content section

This is a set of links to others pages of the simulation editor. These links can be seen as shortcuts allowing a direct access to precise information.

4.3.2. Information page

To be completed.

4.3.3. Source page

This page displays the ADeS simulation file under its XML textual format.

4.4. Simulation instantiation

Introduce more precisely this phase.

4.4.1. Zero snapshot file generation

Once the instantiation of a system succeeded, the *zero snapshot* file is created. This *zero snapshot* is a capture of the simulator's state at the date zero. Any instantiated simulation can then be loaded from that file. For more details about the loading of a simulation, see section 4.5.

As a *zero snapshot* is created for each instantiated simulation, it is possible to instantiate all the simulation of an ADeS project and then load them one by one in any order, without have to re-instantiate them each time.

To be completed.

4.5. Simulation loading

To be completed.

4.6. Simulation execution

To be completed.

4.7. Graphical representation

To be completed.

4.8. Trace of the results

ADeS is able to keep an historic of what happened during simulation: the state changes of components, the performed actions of executable components, etc.

Each created simulation contains a folder named “*result*”. This folder is supposed to encapsulate a file named “*.trace*” into which simulation data are directly written during the simulation. The trace file is written in an XML format.

The file is not supposed to be human-readable since it aims at either building GUI needing an historic of the simulation (chronograms, statistics, etc.) or “playing again” the simulation.

Follows the list of the main XML items that can be found in a trace file:

Item	Description
simu_event	The most generic type of simulation events.
state_change	The state of a component changed.
subcomponent	The subcomponents list changed.
properties	The properties values list changed.
schedulable_elements	The list of schedulable elements changed.

Table 4.1: XML items of a simulation trace file

To be completed.

4.9. Time management

To be completed.

5. TASKS

5.1. Adding a new scheduling policy

This functionality is not yet available.

5.2. Adding a new graphical view

This functionality is not yet available.

6. REFERENCE

6.1. Processor behavior

The behavior of processors is specified by the AADL standard §6.1.

6.2. Thread behavior

The behavior of threads is specified by the AADL standard §5.3.

Additional behavior description may be defined. In the future, ADeS will support the standardized AADL behavior annex currently under definition. For the moment, it defines a simple one (cf. below).

6.3. ADeS behavior annex

ADeS defines a simple behavior annex in order to enrich the default behavior of threads, without involving the complete standardized behavior annex under definition.

6.3.1. Content of the annex

Several basic actions are defined in this annex to represent the behavior of threads:

- `compute(duration)`
consume CPU resource during *duration* to represent the code execution;
- `raise(portName)`
post an event on port *portName*;

The following ones are not yet supported:

- `pend(portName)`
wait for an event on port *portName*;

- `read(dataName)`
read through the data access *dataName*;
- `write(dataName)`
write through the data access *dataName*.

Furthermore, actions list may depend on modes. The syntax is the following one:

```
{ actionsList } in modes (modesList);
```

6.3.2. Example

6.3.2.a. Simple behavior annex

```
thread T
  features
    inPort  : in event port;
    outPort : out event port;
end T ;

thread implementation T.impl
  properties
    Period                => 120ms;
    Compute_Execution_Time => 30ms .. 40ms;
    Dispatch_Protocol     => ( Periodic );
  annex behavior {**
    compute(5ms);
    compute(10ms);
    compute(15ms);
    raise(outPort);
  **};
end T.impl ;
```

6.3.2.b. Behavior annex with modes

```
thread T
  features
    chMode : in event port;
    inPort  : in event port;
    outPort : out event port;
  properties
    Activate_Deadline           => 5ms;
    Activate_Execution_Time     => 5ms .. 5ms;
    Deactivate_Deadline         => 5ms;
    Deactivate_Execution_Time   => 5ms .. 5ms;
    Initialize_Deadline         => 10ms;
    Initialize_Execution_Time   => 10ms .. 10ms;
    Dispatch_Protocol           => ( Periodic );
end T;

thread implementation T.impl
  modes
    normal: initial mode ;
    backup: mode ;
    normal -[ chMode ]-> backup;
    backup -[ chMode ]-> normal;
  properties
    Compute_Execution_Time => 5ms .. 5ms in modes (normal);
    Compute_Execution_Time => 10ms .. 10ms in modes (backup);
    Period                  => 10ms in modes (normal);
```

```
Period => 20ms in modes (backup);  
annex behavior {**  
  {  
    compute(5ms);  
  } in modes (normal);  
  {  
    compute(10ms);  
  } in modes (backup);  
**} ;  
end T.impl;
```


Annexe A. GLOSSARY

A.1. Generic terms

AADL	Architecture analysis and design language
OSATE	Open source AADL tool environment
EMF	Eclipse Modeling Framework
GEF	Graphical Editing Framework

A.2. Terms specific to the project

Annexe B. AADL EXAMPLES

B.1. AADL description of the demo system

```

-- Description of a simple system for the demo.
-- This system executes two threads. One of them is periodic,
-- the second is sporadic and triggered by the first one
-- through an event port.
-----
system DemoSystem
end DemoSystem ;

system implementation DemoSystem.impl
  subcomponents
    singleProcessor : processor DefaultProcessor.impl ;
    singleProcess   : process   DefaultProcess.impl ;
    singleMemory    : memory   DefaultMemory.impl ;
    singleBus       : bus      DefaultBus.impl ;
  properties
    Actual_Memory_Binding    => reference singleMemory
                               applies to singleProcess ;
    Actual_Processor_Binding => reference singleProcessor
                               applies to singleProcess.sender ;
    Actual_Processor_Binding => reference singleProcessor
                               applies to singleProcess.receiver ;
    Actual_Connection_Binding => reference singleBus
                               applies to singleProcess.connection ;
end DemoSystem.impl ;

-- Description of a processor to execute the threads
-----
processor DefaultProcessor
  features
    busAcc : requires bus access DefaultBus.impl ;
end DefaultProcessor ;

processor implementation DefaultProcessor.impl
  subcomponents
    mem : memory DefaultMemory.impl ;
  properties
    Scheduling_Protocol => ( RMS );
end DefaultProcessor.impl ;

-- Description of a process containing two threads
-----
process DefaultProcess
end DefaultProcess ;

process implementation DefaultProcess.impl

```

```

subcomponents
  sender   : thread SenderThread.impl ;
  receiver : thread ReceiverThread.impl ;
connections
  connection : event port sender.outPort -> receiver.inPort ;
end DefaultProcess.impl ;

-- Description of the periodic threads
-----
thread SenderThread
  features
    outPort : out event port;
end SenderThread ;

thread implementation SenderThread.impl
  properties
    Period                => 120ms;
    Compute_Execution_Time => 30ms .. 40ms;
    Dispatch_Protocol     => ( Periodic );
  annex behavior {**
    compute(5ms);
    compute(10ms);
    compute(15ms);
    raise(outPort);
  **};
end SenderThread.impl ;

thread ReceiverThread
  features
    inPort : in event port;
end ReceiverThread ;

thread implementation ReceiverThread.impl
  properties
    Period                => 150ms;
    Compute_Execution_Time => 60ms .. 60ms;
    Dispatch_Protocol     => ( Periodic );
  annex behavior {**
    compute(10ms);
    compute(20ms);
    compute(30ms);
  **};
end ReceiverThread.impl ;

-- Description of the memory onto which the process is bound
-----
memory DefaultMemory
end DefaultMemory;

memory implementation DefaultMemory.impl
end DefaultMemory.impl;

-- Description of the bus onto which connections are bound
-----
bus DefaultBus
end DefaultBus;

bus implementation DefaultBus.impl
  properties
    Propagation_Delay => 5ms .. 5ms;
end DefaultBus.impl ;

```